A_0 = dr(N) — Linear Diophantine Representation Systems

Python Source Code · Bilal el Issaoui · Amsterdam · 2026

DOI: https://doi.org/10.5281/zenodo.20044995

1. General pA + qB (any p ≡ 1 mod q)

Core layer — minimal A, representation count, all representations for any valid pair.

```
# core.py — Layer 1: General version (any pair with p ≡ 1 mod q)
# Author: Bilal el Issaoui, Amsterdam, 2026
# DOI: https://doi.org/10.5281/zenodo.20044995

def a0_general(n, p, q):
    """Minimal A-value: always equals the digital root of N."""
    assert p % q == 1
    return n % q

def r_general(n, p, q):
    """Number of representations of N in the system pA + qB."""
    a0 = n % q
    fm = n // p
    if a0 > fm:
        return 0
    return (fm - a0) // q + 1

def all_representations_general(n, p, q):
    """All non-negative representations (A, B) of N = pA + qB."""
    a0 = n % q
    result = []
    a = a0
    while a <= n // p:
        result.append((a, (n - p * a) // q))
        a += q
    return result
```

2. The 19-9 System

Specific layer — digital root, representation count, all solutions, symmetry point, family boundary G(k).

```
# system_19_9.py — Layer 2: Specific to the 19-9 system
# Author: Bilal el Issaoui, Amsterdam, 2026
# DOI: https://doi.org/10.5281/zenodo.20044995

def digital_root(n):
    """Digital root of N."""
    if n == 0:
        return 0
```

```
    return 1 + (n - 1) % 9

def r(n, p=19, q=9):
    """Number of representations of N in the 19-9 system."""
    a0 = n % q
    fm = n // p
    if a0 > fm:
        return 0
    return (fm - a0) // q + 1

def all_representations(n, p=19, q=9):
    """All non-negative representations (A, B) of N = 19A + 9B."""
    a0 = n % q
    result = []
    a = a0
    while a <= n // p:
        result.append((a, (n - p * a) // q))
        a += q
    return result

def symmetry_point(n, p=19, q=9):
    """Find the symmetry point where dr(A) = dr(B) = dr(N)."""
    r_val = digital_root(n)
    for a, b in all_representations(n, p, q):
        if digital_root(a) == r_val and digital_root(b) == r_val:
            return (a, b)
    return None

def g(k, p=19, q=9):
    """Family boundary: every N >= G(k) has at least k representations."""
    return p * q * (k - 1) + (p - 1) * (q - 1)

# Verification
assert r(958) == 6
assert r(6666) == 39
assert symmetry_point(958) == (40, 22)
assert symmetry_point(6666) == (24, 690)
assert g(10) == 1683
```

---

3. Matryoshka Chain

Complete Matryoshka calculator — links every value in the reduction chain N → S(N) → dr(N) to its A-coordinate.

```
# matryoshka.py — Layer 3: Complete Matryoshka calculator
# Author: Bilal el Issaoui, Amsterdam, 2026
# DOI: https://doi.org/10.5281/zenodo.20044995
```

```
def digit_sum(n):
    """Sum of all digits of N."""
    return sum(int(c) for c in str(n))

def digital_root(n):
    """Digital root of N."""
    while n >= 10:
        n = digit_sum(n)
    return n

def reduction_chain(n):
    """Complete reduction chain N -> S(N) -> ... -> dr(N)."""
    chain = [n]
    current = n
    seen = {n}
    while current >= 10:
        nxt = digit_sum(current)
        if nxt not in seen:
            chain.append(nxt)
            seen.add(nxt)
        current = nxt
    return chain

def all_representations(n, p=19, q=9):
    """All non-negative representations (A, B) of N = pA + qB."""
    if n < 0:
        return []
    a0 = n % q
    result = []
    a = a0
    while a <= n // p:
        b = (n - p * a) // q
        if p * a + q * b == n and b >= 0:
            result.append((a, b))
        a += q
    return result

def link_chain_to_representations(n, p=19, q=9):
    """Link every value in the reduction chain to its A-coordinate."""
    chain = reduction_chain(n)
    reps = all_representations(n, p, q)
    a_values = {a for a, _ in reps}
    dr_n = digital_root(n)
    coupling = []
    for ci in chain:
        if ci in a_values:
            b = next(b for a, b in reps if a == ci)
```

```
            ver = p * ci + q * b
            coupling.append({
                'ci': ci, 'present': True, 'B': b,
                'verification': ver, 'correct': ver == n,
            })
        else:
            coupling.append({
                'ci': ci, 'present': False, 'B': None,
                'verification': None, 'correct': False,
            })
    symmetry_points = [
        (a, b) for a, b in reps
        if digital_root(a) == dr_n and digital_root(b) == dr_n
    ]
    a0 = n % q
    r_formula = (n // p - a0) // q + 1 if a0 <= n // p else 0
    return {
        'N': n, 'p': p, 'q': q, 'dr_N': dr_n, 'A0': a0,
        'chain': chain,
        'chain_str': ' -> '.join(str(c) for c in chain),
        'R': len(reps), 'R_formula': r_formula,
        'representations': reps, 'coupling': coupling,
        'symmetry_points': symmetry_points,
    }

# Verification
result = link_chain_to_representations(958)
assert result['dr_N'] == 4
assert result['A0'] == 4
assert result['R'] == 6
assert result['symmetry_points'] == [(40, 22)]
```

---

4. Sigma R_4 — 4D J-block implementation

Full 4D representation count via J-block structure. 49 KB — see GitHub for complete source.

The complete implementation is available at:
https://github.com/A19dammer91/N-pA-qB-with-p-1-mod-q-extended-to-3D-and-4D-systems/blob/main/Sigma_R%E2%82%84.py

Key functions included in the full file:

```
    floor_sum(n, m, a, b) — O(log m) floor sum via AtCoder Library reduction
    sum_mod_sequence(C, r, q, n_start, n_end) — O(log q) mod sequence sum
    sum_floor_i_div_k(m, k) — O(1) closed-form K-sum
    r3_kernel(N, p, q, r) — O(1) 3D representation count
    r4_jblock(N, p, q, r, s) — O(N/(p*r)) 4D count via J-blocks
    verify_r4(N_max, p, q, r, s) — Verification suite vs brute force
```

Zenodo: https://doi.org/10.5281/zenodo.19474707 | GitHub:
github.com/A19dammer91/N-pA-qB-with-p-1-mod-q-extended-to-3D-and-4D-systems